

An Independent Billing Audit of Claude’s Production API: 4 Experiments Across Haiku, Sonnet, and Opus Confirm Mathematical Honesty; Tool-Definition Overhead and Silent Desktop-App Routing Remain Consumer Concerns

Chen, Ho Yiing¹

¹Independent researcher, charenix.com, ORCID: 0009-0006-6816-9891
abbychen981008@gmail.com

2026-05-25

Abstract

I run an independent inspection of Claude’s production billing API across four controlled experiments. (1) Hidden-overhead probe: across prompts ranging from 50 to 15,000 BPE units, the `messages/count_tokens` endpoint returns exactly the same input-unit count as `messages/create` reports as `usage.input_tokens`, with zero delta on every sample. (2) Definition inflation: a single tool definition adds 571 extra input tokens on a prompt that would otherwise cost 14 units; 20 added definitions add 1,996 extra pieces per request. (3) Stored-prefix accounting consistency: a 2,052-unit cacheable prefix written on call 1 is reported as exactly 2,052 `cache_read_input_tokens` on call 2, with a write-to-read ratio of 1.0000 to four decimal places. (4) Extended-thinking transparency: when Opus is invoked with `thinking.budget_tokens = 2048`, the additional output values reported in `usage` match the visible reasoning content returned to the caller. Four experiments. Four clean results. All four find that the API-level billing arithmetic is honest. The consumer concerns surfaced by this work are not fraud but: (a) measurable, undocumented definition overhead that every Claude Code and MCP-server user pays per request; and (b) silent routing of paid subscription users to pay-as-you-go API billing via the desktop application’s automatically-created coworking API key, on top of the subscription. The first is a transparency issue, the second is a consent issue. Neither is a math issue. Open methodology, raw JSON results, and source code at the Mercury paper handoff repository [1].

1 Introduction

Subscriber complaints about AI provider billing are widespread and largely anecdotal. The accusation that providers silently pad counts or misreport stored savings has been repeated in social media, support tickets, and the occasional news article. To date, no published controlled inspection has measured Claude’s reported counts against an independently-verifiable ground truth.

This paper presents such a study. The method rests on a single design choice: the provider exposes both a `messages/count_tokens` endpoint [3] (free, returns the tokenizer’s count for a given message structure) and the standard `messages/create` endpoint (returns `usage.input_tokens` after actual

model invocation). If the provider pads, inflates, or misreports, the gap between these two endpoints should reveal it.

Four experiments probe four distinct forms of suspected inflation: hidden constant overhead, definition inflation, cache-accounting fraud, and reasoning-unit transparency. The model coverage spans Haiku, Sonnet, and Opus (Claude 4.5/4.7 generation).

2 Method

2.1 Ground-truth comparator

For each input message structure M , the comparison applies two numbers:

$$N_{\text{count}}(M) = \text{messages.count_tokens}(M).\text{input_tokens}$$

$$N_{\text{actual}}(M) = \text{messages.create}(M).\text{usage.input_tokens}$$

A delta $\Delta = N_{\text{actual}} - N_{\text{count}}$ of zero indicates the billed count exactly matches the tokenizer’s count for the same payload. A positive constant Δ indicates a constant overhead (potentially a hidden system prompt). A Δ that scales with $|M|$ indicates proportional padding (the most concerning scenario).

2.2 Controls

All requests use deterministic decoding (temperature unspecified, so default 1.0 with `max_tokens` bounded). No streaming. The same client (Python anthropic SDK v0.104.1) is used for all calls. Time between consecutive calls in cache-related experiments is held to 3 seconds, well within the documented 5-minute and 1-hour TTL windows [2]. The work was conducted on 2026-05-25 from a consumer machine in Taiwan.

2.3 Experiment design

E1 (hidden overhead) sends five prompts of nominal sizes 50, 500, 2000, 5000, and 15000 units, constructed as repetitions of the fixed string ```The quick brown fox jumps over the lazy dog. ```. For each, both the two counters are recorded. Model: `claude-haiku-4-5`.

E2 (definition inflation) fixes the user prompt to ```What is the capital of France?``` and varies the number of supplied entries across $\{0, 1, 5, 20\}$. Each entry is a synthetic specification with a name, a one-line description, and a JSON-schema input with two properties. Model: `claude-haiku-4-5`.

E3 (cache accounting) sends the same message twice. The message contains a 2,052-unit cacheable prefix (marked with `cache_control: ephemeral`) and a short instruction suffix. Call 1 establishes the cache; call 2 reads from it. The comparison checks the equality $N_{\text{cache_read}, \text{call 2}} = N_{\text{cache_creation}, \text{call 1}}$. Model: `claude-sonnet-4-5`.

E4 (reasoning transparency) invokes Opus on a multi-step math word problem twice: once without thinking, once with `thinking.budget_tokens = 2048`. The comparison checks the reported `output_tokens` delta against the visible thinking content length. Model: `claude-opus-4-5`.

3 Results

3.1 E1: zero hidden overhead

The reported N_{actual} matches N_{count} to the byte on every sample. The model adds no hidden system prompt, no proportional padding, no per-request inflation that would survive past the `count_tokens` ground truth.

Target	N_{count}	N_{actual}	Δ
50	63	63	0
500	558	558	0
2000	2208	2208	0
5000	5508	5508	0
15000	16508	16508	0

Table 1: Across 5 prompts spanning 50 to 15,000 units, the count_tokens path and the actual billed input value agree exactly. No hidden constant overhead. No scaling padding.

n_{tools}	N_{actual}	Δ from baseline	per-entry marginal
0	14	0	—
1	585	571	571
5	885	871	217
20	2010	1996	100

Table 2: Tool definitions add measurable but reported input overhead. The first entry incurs the dominant fixed cost (the definition-handling system template). Subsequent entries add 75 to 100 pieces each.

3.2 E2: measurable definition overhead

The first definition adds 571 BPE pieces of overhead; this likely reflects the platform’s definition-handling system prompt template, which is injected once when any definition is present. Each additional entry adds approximately 100 pieces. This overhead is fully reported in usage.input_tokens: it is not fraud, but it is undocumented in public pricing. A subscriber with a 20-entry Claude Code or MCP configuration pays ~2,000 input pieces per request regardless of message content. Over 1,000 daily messages, this is 2 million items of pure definition overhead per day.

3.3 E3: cache accounting is bit-accurate

Call	input	cache_creation	cache_read
Call 1 (writer)	11	2052	0
Call 2 (reader)	11	0	2052

Table 3: The cacheable prefix written on call 1 is reported as exactly the same number of read entries on call 2. Write-to-read ratio: 1.0000.

The cache write on call 1 (2,052 items) is recovered exactly as the cache read on call 2 (2,052 items). There is no shrinkage, no inflation, no off-by-one. The stored discount the subscriber is told they receive (10% of input price for cache-read entries) is computed against the exact same count that was written. Subscribers who fear that the platform silently shrinks their reported stored hits to inflate effective price are, on this evidence, mistaken.

3.4 E4: reasoning pieces are visible to the caller

Output items with extended thinking enabled match the sum of visible thinking content plus the visible text reply, within tokenizer granularity. The work does not detect “invisible” reasoning items being billed.

Config	input	output	notes
no thinking	47	359	visible text only
thinking, 2048 budget	76	787	thinking_chars=1385 + text_chars=736

Table 4: Extended-thinking mode adds 29 items to input (the thinking system prompt addition) and increases output from 359 to 787. The visible thinking content (1385 chars) plus text response (736 chars) totals 2121 chars, which at typical English BPE rates of ~ 2.7 chars per item gives roughly 786 items, matching the reported 787. The reasoning chunks billed correspond to content the caller can read.

4 Discussion

4.1 API-level arithmetic vs UX-level deception

Headline. The production API performs its accounting honestly. The four most common forms of suspected fraud (hidden padding, stored shrinkage, definition inflation beyond what’s reported, secret reasoning items) all fail to materialise under controlled measurement. Independent researchers and litigation teams who are looking for fraud at the API layer will not find it here.

This does not mean subscribers have no grievances. Two real consumer-impact issues survive this work:

Definition overhead is large and undocumented. A 20-entry Claude Code configuration costs roughly 2,000 input items per request before the operator types a single character. This is fully reported in usage. `input_tokens` (and is therefore not fraud), but it is not surfaced in the public pricing copy or in the desktop app’s UI. Subscribers routinely add MCP servers and Skill plugins without realising each one inflates the per-request floor.

The desktop application’s automatic API key creation silently routes subscription users to pay-as-you-go billing. The Cowork desktop application can create an API key labelled coworking on the user’s account during installation, and route subsequent agent traffic through that key rather than the existing Pro or Max subscription credit. This author personally experienced USD \$89.90 of unexpected pay-as-you-go charges over an 18-day period while concurrently holding a paid monthly Max subscription (USD \$200). No step in the desktop app’s onboarding informed the buyer that a separate API-billed key would be created and used. This is not fraud (every charge is correctly computed against the actual usage), but it is a routing surprise that is, on the evidence of public support-channel responses, treated by the platform as “legitimate usage” rather than a UX defect.

These two issues are not exposed by the controlled experiments because they are not math problems. They are transparency and consent issues. A fair public account of Claude’s billing should therefore distinguish “the numbers are right” (true, on this work) from “the buyer knew what they were buying” (questionable, on the desktop-app routing question).

4.2 Why `count_tokens` makes this kind of inspection possible

A brief aside on why `count_tokens` makes this kind of investigation possible at all: the choice by the provider to ship a free, low-latency tokenizer service is unusual among major LLM platforms, who often reserve the tokenizer for their own internal billing pipeline rather than expose it as a queryable interface. OpenAI for example publishes `tiktoken` as a Python package, which is an even stronger move (the caller does not even need to call the API), but lacks the per-account, per-tool, per-cache-config awareness the live endpoint can provide. Google’s Gemini API offers a similar count-style service with comparable semantics. Mistral’s API does not. The result is that an independent assessment of the form presented here is feasible against the two listed but not symmetrically against every major firm. This asymmetry is itself a finding worth reporting: provider billing accountability is gated, in practice, by whether the company chose to ship a publicly-queryable tokenizer interface at all. Subscribers who suspect billing irregularities at a supplier that does not offer such an interface have, by construction, no way to verify

their suspicion against ground truth, and must rely on platform goodwill (or litigation discovery) for the same level of certainty. The published sparse-autoencoder work on Claude internals [4] suggests the company has substantial internal transparency, but the public-facing tokenizer endpoint is what enables third parties to do this kind of audit at all.

5 Limitations

Single point in time. The investigation was conducted on 2026-05-25.

Model coverage. The probe covers Haiku, Sonnet, and Opus at the 4.5/4.7 generation. Older models, Claude 3.x, and yet-unreleased variants are not measured.

Single API region. All requests went to the default Anthropic endpoint from a Taiwan client. Regional billing variations (if any exist) are not measured.

Small sample per experiment. E1 uses five prompt sizes, E2 four entry counts, E3 a single prompt pair, E4 a single problem with two configurations.

No third-party tokenizer comparison. The work uses the provider’s own `count_tokens` endpoint as ground truth. If that endpoint itself were biased, the analysis would not detect it.

One-line summary

Math honest. UX not.

6 Conclusion

Four experiments. Four clean results. Production API arithmetic for billing is honest. Hidden overhead is zero. Definition overhead is large but fully reported. Stored prefix-to-read ratio is 1.0000. Extended-thinking output values match visible content. Subscribers with arithmetic-fraud suspicions should look elsewhere. Subscribers with UX-and-consent grievances (in particular, the desktop application’s automatic coworking API key routing) have legitimate concerns that this work’s data does not address but also does not refute.

The Mercury paper handoff repository [1] hosts the open source and raw results.

References

- [1] Chen, Ho Yiing. Mercury paper handoff: drafts, analysis scripts, and Tier-A / Tier-B / Tier-C observation data. <https://github.com/norika1207-lab/mercury-paper-handoff>, 2026.
- [2] Anthropic. Prompt caching documentation. <https://docs.anthropic.com/en/docs/build-with-claude/prompt-caching>, accessed 2026-05-25.
- [3] Anthropic. Messages: Count tokens (API reference). <https://docs.anthropic.com/en/api/messages-count-tokens>, accessed 2026-05-25.
- [4] Bricken, T., Templeton, A., Batson, J., Chen, B., Jermyn, A., Conerly, T., Turner, N., Anil, C., Denison, C., Askeel, A., Lasenby, R., Wu, Y., Kravec, S., Schiefer, N., Maxwell, T., Joseph, N., Tamkin, A., Nguyen, K., McLean, B., Burke, J., Hume, T., Carter, S., Henighan, T., and Olah, C. Towards Monosemanticity: Decomposing Language Models With Dictionary Learning. Transformer Circuits Thread, Anthropic, October 2023. <https://transformer-circuits.pub/2023/monosemantic-features>.